IN THE CLAIMS:

Please amend the claims as follows:

1. (Currently Amended) A method of coordinating access to common memory by multiple program threads comprising the steps of:

in each given program thread,

(a) detecting the beginning of a critical section of the given program thread in which ~~conflicts~~ interruption to access of the common memory could occur resulting from execution of other program threads;

(b) speculatively executing the critical section; and

(c) committing the speculative execution of the critical section if there has been no ~~conflict~~ interruption to access of the common memory and squashing the speculative execution of the critical section if there has been an ~~conflict~~interruption.

2. (Currently Amended) The method of claim 1 wherein the ~~conflict~~ interruption is:

(a) another thread writing data read by the given program thread in the critical section, or

(b) another thread reading or writing data written by the given program thread.

3. (Currently Amended) The method of claim 2 wherein the ~~conflict~~ interruption is detected by an invalidation of a cache block holding data of the critical section.

4. (Original) The method of claim 1 wherein the speculative execution is committed at the end of the critical section.

5. (Original) The method of claim 4 wherein the end of the critical section is detected by a pattern of instructions typically associated with a lock release.

6. (Original) The method of claim 5 wherein the pattern of instructions is a store instruction to a deduced lock variable address.

7. (Original) The method of claim 1 wherein the speculative execution is committed at a resource boundary limiting further speculation.

8. (Currently Amended) The method of claim 7 including the step of:

(d) if at step (c) there was no ~~conflict~~ interruption from the execution of another thread acquiring a lock variable allowing the given thread to have exclusive access to the critical section and continuing execution from the commitment point to the conclusion of the critical section.

9. (Original) The method of claim 1 wherein the speculative execution is committed upon the occurrence of a non cacheable operation limiting further speculation.

10. (Currently Amended) The method of claim 9 including the step of:

(d) if at step (c) there was no ~~conflict~~ interruption from the execution of another thread acquiring a lock variable allowing the given thread to have exclusive access to the critical section and continuing execution from the commitment point to the conclusion of the critical section.

11. (Original) The method of claim 1 wherein step (a) includes reading of a lock variable and wherein step (b) is performed only when the lock variable is not held by another program thread.

12. (Original) The method of claim 1 wherein step (a) includes reading a prediction table holding historical data indicating past successes in speculatively executing the critical section and wherein step (b) is performed only when the prediction table indicates a likelihood of successful speculative execution of the critical section of above a predetermined threshold.

13. (Original) The method of claim 1 wherein step (a) deduces the beginning of a critical section by detecting patterns of instructions typically associated with a lock acquisitions.

14. (Original) The method of claim 13 wherein the pattern includes an atomic read/modify/write sequence.

15. (Original) The method of claim 1 wherein the critical section is preceded by a lock acquisition section and including the step of eliding the lock acquisition before step (b).

16. (Currently Amended) The method of claim 1 wherein the critical section ends with a lock release section and including the step of eliding the lock release section after step (c) when at step (c) upon reaching the end of the critical section, no ~~conflict~~ interruption from the execution of another thread occurred.

17. (Currently Amended) The method of claim 1 including the further step of:

(d) after squashing the speculative execution of the critical section if there has been a ~~conflict~~interruption, re-executing the critical section speculatively.

18. (Currently Amended) The method of claim 17 wherein the speculative re-execution of the critical section is repeated up to a predetermined number of times until there is not a ~~conflict~~interruption.

~~18~~19. (Currently Amended) The method of claim 17 wherein (d) if after the predetermined number of tries there remains a ~~conflict~~ interruption from the execution of another thread, acquiring a lock variable allowing the given thread to have exclusive access to the critical section and continuing execution of the critical section from its beginning.

20. (Original) The method of claim 1 wherein the speculation executes the critical section using a cache memory to record the speculative execution without visibility to other processing units.

21. (Original) The method of claim 1 wherein the speculation executes the critical section eliding write instructions that do not change a value of memory location being written to.

22. (Currently Amended) A lock elision circuit for a computer architecture allowing the access of common memory by multiple program threads, the circuit comprising:

means for controlling the execution of each given program thread to:

(a) detect the beginning of a critical section of the given program thread in which ~~conflicts to~~ unpredicted access of the common memory ~~could occur resulting from execution of~~ by other program threads could interfere with the given program thread;

(b) speculatively execute the critical section; and

(c) commit the speculative execution of the critical section if there has been no ~~conflict~~ interfering access to the common memory by other program threads and squashing the speculative execution of the critical section if there has been an ~~conflict~~interfering access to the common memory by other program threads.

23. (Currently Amended) The lock elision circuit of claim 22 wherein the ~~conflict~~ interfering access to the common memory is:

(a) another thread writing data read by the given program thread in the critical section, or

(b) another thread reading or writing data written by the given program thread.

24. (Currently Amended) The lock elision circuit of claim 23 wherein the computer architecture includes a cache and the ~~conflict~~ interfering access to the common memory is detected by an invalidation of a cache block holding data of the critical section.

25. (Original) The lock elision circuit of claim 22 wherein the speculative execution is committed at the end of the critical section.

26. (Original) The lock elision circuit of claim 25 wherein the end of the critical section is detected by a pattern of instructions typically associated with a lock release.

27. (Original) The lock elision circuit of claim 26 wherein the pattern of instructions is a store instruction to a deduced lock variable address.

28. (Original) The lock elision circuit of claim 22 wherein the speculative execution is committed at a resource boundary limiting further speculation.

29. (Currently Amended) The lock elision circuit of claim 28 wherein when there is no ~~conflict~~ interfering access to the common memory from the execution of another thread acquiring a lock variable, the lock elision circuit allows the given thread to have exclusive access to the critical section and continues execution from the commitment point to the conclusion of the critical section.

30. (Original) The lock elision circuit of claim 22 wherein the lock elision circuit reads a lock variable and speculatively executes the critical section only when the lock variable is not held by another program thread.

31. (Original) The lock elision circuit of claim 22 wherein the speculative execution is committed upon the occurrence of a non cacheable operation limiting further speculation.

32. (Currently Amended) The lock elision circuit of claim 31 including the step of:

(d) if at step (c) there was no ~~conflict~~ interfering access to the common memory from the execution of another thread acquiring a lock variable allowing the given thread to have exclusive access to the critical section and continuing execution from the commitment point to the conclusion of the critical section.

33. (Original) The lock elision circuit of claim 22 including a prediction table holding historical data indicating past successes in speculatively executing the critical section and wherein the lock elision circuit speculatively executes the critical section only when the prediction table indicates a likelihood of successful speculative execution of the critical section of above a predetermined threshold.

34. (Original) The lock elision circuit of claim 22 wherein the lock elision circuit determines the beginning of a critical section by detecting patterns of instructions typically associated with a lock acquisitions.

35. (Original) The lock elision circuit of claim 34 wherein the pattern includes an atomic read/modify/write sequence.

36. (Original) The lock elision circuit of claim 22 wherein the critical section is preceded by a lock acquisition section and wherein the lock elision circuit elides the lock acquisition before speculation.

37. (Currently Amended) The lock elision circuit of claim 22 wherein the critical section ends with a lock release section and wherein the lock elision circuit elides the lock release section after speculation when upon reaching the end of the critical section, no ~~conflict~~ interfering access to the common memory from the execution of another thread occurred.

38. (Currently Amended) The lock elision circuit of claim 22 wherein after squashing the speculative execution of the critical section, if there has been an ~~conflict~~interfering access to the common memory, the lock elision circuit re-executes the critical section speculatively.

39. (Currently Amended) The lock elision circuit of claim 38 wherein the lock elision circuit repeats the speculative re-execution of the critical section up to a predetermined number of times until there is not an ~~conflict~~interfering access to the common memory.

40. (Currently Amended) The lock elision circuit of claim 39 wherein if after the predetermined number of tries there remains an ~~conflict~~ interfering access to the common memory from the execution of another thread, the lock elision circuit allows acquisition of a lock variable allowing the given thread to have exclusive access to the critical section and continuing execution of the critical section from its beginning.

41. (Original) The lock elision circuit of claim 22 wherein the computer architecture includes a cache memory and the lock elision circuit uses the cache memory to record the speculative execution without visibility to other processing units.

42. (Original) The lock elision circuit of claim 22 wherein the lock elision circuit elides write instructions within the critical section that do not change a value of memory location being written to.

43. (Currently Amended) A method of coordinating the execution of multiple program threads executing critical sections of a program, the critical sections accessing common memory and being preceded by lock acquisition sections where writing to a lock variable mediates exclusive access to the common memory, the method comprising the steps of:

for each given thread:

(a) upon reaching a lock acquisition section, reading the lock variable to see if the lock has been acquired by another thread; ~~and~~

(b) when at step (a) the lock has not been acquired by another thread, allowing the given thread to complete execution of the critical section without acquiring permission to write to the lock variable; and

(c) wherein upon complete execution of the critical section without acquiring permission to write to the lock variable, if another thread has acquired the lock, squashing the execution of the critical section and then re-executing the critical section.

44. (Original) The method of claim 43 wherein the lock variable is held within a cache and wherein acquiring permission to write to the lock variable includes the step of obtaining ownership of at least a portion of the cache holding the lock variable.

45. (Currently Amended) The method of claim 43 wherein during step (b) the execution is not committed and ~~including the further~~ at step (c) the lock was not acquired by another thread, performing the step of:

(~~c~~d) committing the execution at the completion of the critical section~~, only if the lock has not be acquired by another thread during that execution~~.

46. (Original) The method of claim 45 wherein acquisition of the lock by another thread is detected by a cache protocol messages.

47. (Currently Amended) The method of claim 43 wherein during step (b) the execution is not committed and including the further step of:

(~~c~~d) when during step (b) the resources required for execution without commitment are exhausted, acquiring the lock, committing the execution and continuing with execution until completion of the critical section.